CS 61A Structure and Interpretation of Computer Programs
Spring 2022

Midterm 1 Solutions

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

(a) What is your full name?

(b) What is your student ID number?

1. (7.0 points) What Would Python Dip?

Assume the following code has been executed.

```
def dipping(dots):
    if print("you dip"):
        return print("i dip")
    else:
        return print(dots) or dots or print("we dip")
```

What would the Python interpreter display? If the interpreter would include a new line, please enter a new line in your answer.

(a) (2.0 pt)

```
>>> dipping(0)
```

```
you dip 0 we dip
```

(b) (2.0 pt)

```
>>> dipping(555)
```

```
you dip 555 555
```

(c) (1.0 pt) What is the return value of `dipping(0)`?

None

(d) (1.0 pt) What is the return value of `dipping(-666)`?

-666

(e) (1.0 pt) Rewrite the body of the `dipping` function in a single line of code.

```
def dipping(dots):
    ---------------------
```

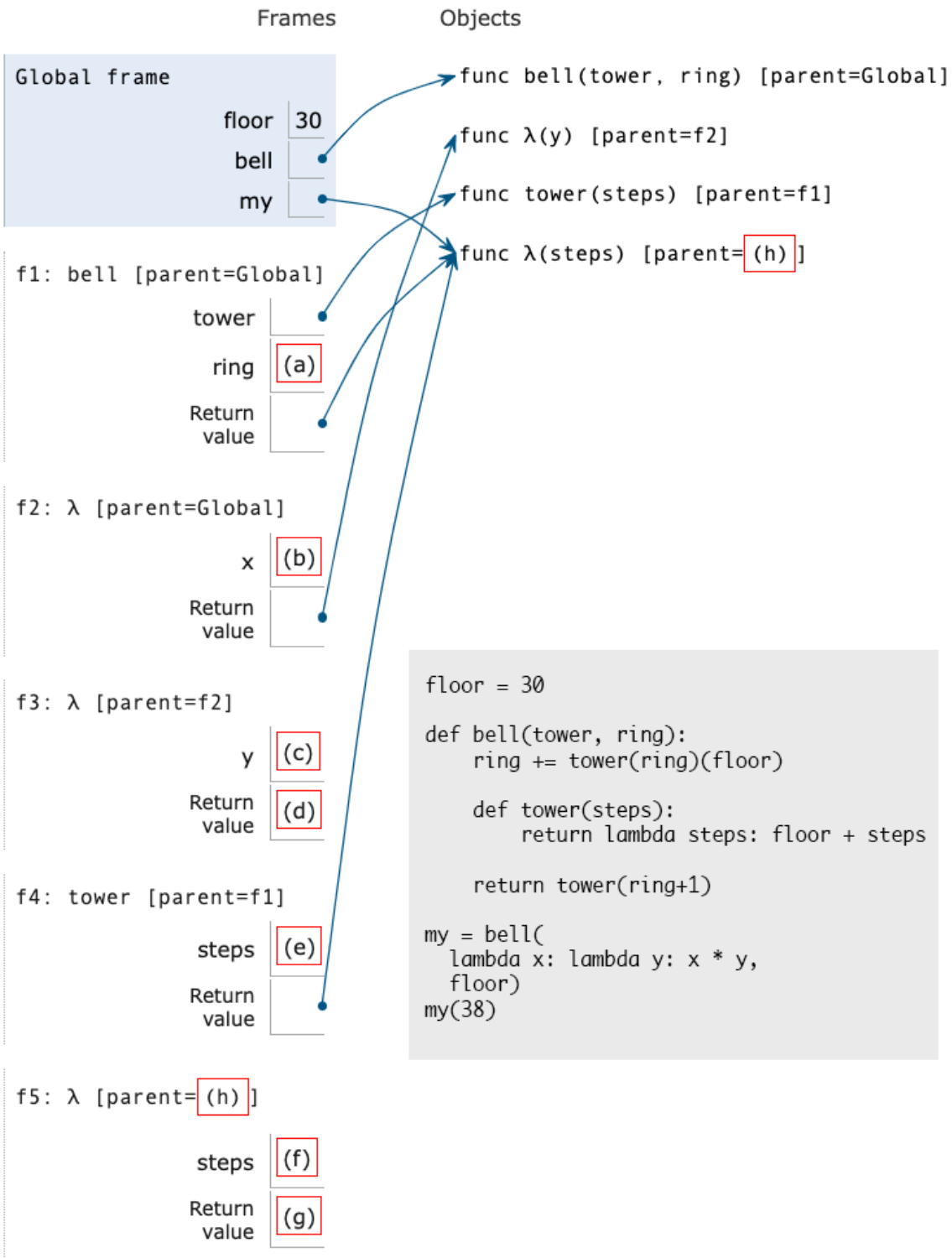
```
return print("i dip") if print("you dip") else (print(dots) or dots or
print("we dip"))
```

2. (8.0 points) Ring My Bell Tower

The following code was used to generate the environment diagram below:

```
floor = 30

def bell(tower, ring):
    ring += tower(ring)(floor)

    def tower(steps):
        return lambda steps: floor + steps

    return tower(ring+1)

my = bell(lambda x: lambda y: x * y, floor)
my(38)
```

The environment diagram below represents the *final* state of the environment. The code is also provided to the right of the diagram, for convenience. Line numbers have been omitted intentionally.



(a) (1.0 pt) Fill in blank (a)

930

(b) **(1.0 pt)** Fill in blank (b)

30

(c) **(1.0 pt)** Fill in blank (c)

30

(d) **(1.0 pt)** Fill in blank (d)

900

(e) **(1.0 pt)** Fill in blank (e)

931

(f) **(1.0 pt)** Fill in blank (f)

38

(g) **(1.0 pt)** Fill in blank (g)

68

(h) **(1.0 pt)** Fill in blank (h)

f4

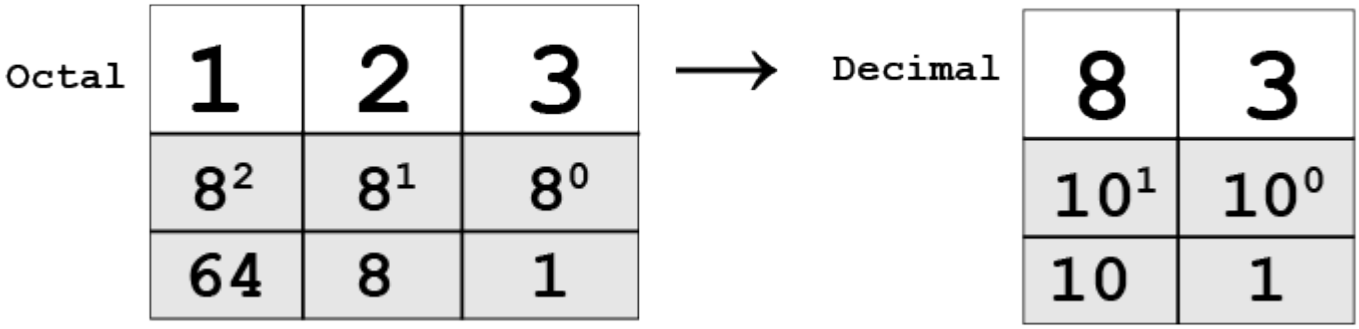
3. (6.0 points) Doctor Octalpus, Reborn

The standard number representation system is the decimal system, where each digit in a number represents a power of ten. The right-most digit is the ones' place, the next digit is the tens' place, etc.

In the octal system, each digit in a number represents a power of eight. The right-most digit is still the 1's place, but the next digit is the 8's place, the next digit is the 64's place, etc. Each digit ranges from 0-7, so octal numbers will never contain the digits 8 or 9.

To convert a number represented in octal to a number represented in decimal, each digit must be multipled by the appropriate power of eight. For example, `123` is actually `(1 * 64) + (2 * 8) + (3 * 1)`, resulting in a decimal representation of 83.

The diagram visualizes the equivalence between the octal and decimal numbers:



Implement `convert_to_decimal`, which takes an octal number and returns the decimal equivalent. The octal number will always start with a non-0 digit, and the number will always be positive.

```
def convert_to_decimal(octal):
    """
    >>> convert_to_decimal(3)   # (8^0 * 3)
    3
    >>> convert_to_decimal(23)  # (8^1 * 2) + (8^0 * 3)
    19
    >>> convert_to_decimal(123) # (8^2 * 1) + (8^1 * 2) + (8^0 * 3)
    83
    """
    decimal = 0
    curr_place = _________
                     (a)
    _________:
       (b)
        curr_digit = _________
                         (c)
        decimal = _________
                      (d)
        curr_place = _________
                         (e)
        octal = _________
                    (f)
    return decimal
```

(a) (1.0 pt) Fill in blank (a).

```
1
```

(b) (1.0 pt) Fill in blank (b).

```
while octal > 0
```

(c) (1.0 pt) Fill in blank (c).

```
octal % 10
```

(d) (1.0 pt) Fill in blank (d).

```
decimal + (curr_digit * curr_place)
```

(e) (1.0 pt) Fill in blank (e).

```
curr_place * 8
```

(f) (1.0 pt) Fill in blank (f).

```
octal // 10
```

4. (6.0 points) Forbidden Digits

Implement `forbid_digit`, a higher-order function which takes two arguments, a function `f` and a digit `forbidden`, and returns another function. If the returned function is passed a number where the digit in the 1s place is equal to the forbidden digit, it should return the result of calling the given function on the number without that final digit. Otherwise, it should return the result of calling the given function on the number.

```
def forbid_digit(f, forbidden):
    """
    >>> g = forbid_digit(lambda y: 200 // (y % 10), 0)
    >>> g(11)
    200
    >>> g(10)
    200
    >>> g = forbid_digit(lambda x: f'{x}a', 6)
    >>> g(61)
    '61a'
    >>> g(66)
    '6a'
    >>> g = forbid_digit(g, 3)
    >>> g(43)
    '4a'
    >>> g(63)
    '0a'
    >>> g(44)
    '44a'
    """
    def forbid_wrapper(n):
        if ________:
              (a)
            --------
              (b)
        else:
            --------
              (c)
    --------
       (d)
```

(a) (1.0 pt) Fill in blank (a).

```
n % 10 == forbidden
```

(b) (1.0 pt) Fill in blank (b).

```
return f(n // 10)
```

(c) (1.0 pt) Fill in blank (c).

```
return f(n)
```

(d) **(1.0 pt)** Fill in blank (d).

```
return forbid_wrapper
```

(e) **(2.0 pt)** Rewrite the body of the `forbid_digit` function in a single line of code.

```
def forbid_digit(f, forbidden):
    ____________________
```

```
return lambda n: f(n//10) if n % 10 == forbidden else f(n)
```

5. (7.0 points) The Floor is Lava

Implement `lava_hopper`, a function that "hops" from one number to the next computed number and tries to avoid any number detected as "lava". When it does land on "lava", it steps backwards by one number until it finds a non-lava number and then keeps hopping.

The function takes four arguments: `start_number` (the initial number), `goal_number` (the target number), `next_hop` (a function that computes the next number based on the current), and `is_lava` (a function that returns a boolean indicating if a number is lava), and it returns the minimum number of hops required to get from `start_number` to at least `goal_number`. The number of hops does *not* include steps backwards. If either the `start_number` or `goal_number` spots are lava, it returns the string 'No lava allowed there!'.

For example, consider this call

```
lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 4)
```

The function starts from the number 1 and then hops to the numbers 2, 4, realizes that's lava, steps back to 3, hops to 6, hops to 12, and returns 4 (the number of hops required to get to/past 8).

Note that depending on the functions passed in for `next_hop` and `is_lava`, it is possible for a correct `lava_hopper` implementation to result in an infinite loop.

```
def lava_hopper(start_number, goal_number, next_hop, is_lava):
    """
    >>> # hops from 1->2, 2->4, 4->8
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: False)
    3
    >>> # hops from 1->2, 2->4, steps to 3, hops 3->6, hops 6->12
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 4)
    4
    >>> # hops from 1->2, 2->4, 4->8, steps to 7, then 6, then 5, hops to 10
    >>> lava_hopper(1, 10, lambda x: x * 2, lambda x: 6 <= x <= 8)
    4
    >>> # hops from 3->6, 6->12, steps to 11, hops 11->22
    >>> lava_hopper(3, 20, lambda x: x * 2, lambda x: x % 10 == 2)
    3
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 1)
    'No lava allowed there!'
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 8)
    'No lava allowed there!'
    """
    if __________:
           (a)
        return 'No lava allowed there!'
    num_hops = 0
    while __________:
              (b)
        ________________:
              (c)
            ----------
                (d)
        start_number = __________
                    (e)
        ----------
            (f)
    return num_hops
```

(a) (1.0 pt) Fill in blank (a).

```
is_lava(start_number) or is_lava(goal_number)
```

(b) (1.0 pt) Fill in blank (b).

```
start_number < goal_number
```

(c) (1.0 pt) Fill in blank (c).

```
while is_lava(start_number)
```

(d) (1.0 pt) Fill in blank (d).

```
start_number -= 1
```

(e) (1.0 pt) Fill in blank (e).

```
next_hop(start_number)
```

(f) (1.0 pt) Fill in blank (f).

```
num_hops += 1
```

(g) (1.0 pt) Write a call to `lava_hopper` that would result in an infinite loop.

```
lava_hopper(______________________)
```

```
1, 8, lambda x: x + 1, lambda x: x == 6
```

6. (6.0 points) Curry Up Now

The function `order_meal` takes three arguments, `item_price`, `item_quantity`, and `ordered_at`, and either returns the total cost of the meal or reports how many hours should be waited before ordering. Only the doctests are shown below, as the implementation is not necessary for completing the question.

```
def order_meal(item_price, item_quantity, ordered_at):
    """
    >>> order_meal(5.99, 5, 11)
    29.95
    >>> order_meal(9.99, 5, 20)
    49.95
    >>> order_meal(8.99, 5, 7)
    'Wait!'
    """
    # Code intentionally omitted
```

Implement `curry_up_now`, a function that curries `order_meal` into a chain of three functions that each take a single argument. Once the third function is called, it should attempt to order the meal and print out the result. If the meal was successfully ordered during business hours, it should then return another curried function that can re-order the same item with a 50% discount.

```
def curry_up_now(item_price):
    """
    >>> curry_up_now(2.99)(2)(15)
    5.98
    <function <lambda>>
    >>> lunch_special = curry_up_now(8.99)
    >>> lunch_special(5)(11)
    44.95
    <function <lambda>>
    >>> lunch_special(3)(13)(2)(14)
    26.97
    8.99
    >>> no_discount = curry_up_now(10.99)(4)(7)
    Wait!
    >>> print(no_discount)
    None
    """
    def order_quantity(item_quantity):
        def by(ordered_at):
            result =  ________
                        (a)
            ________
               (b)
            ________:
               (c)
                return ________
                          (d)
        return by
    return order_quantity
```

(a) (1.0 pt) Fill in blank (a).

```
order_meal(item_price, item_quantity, ordered_at)
```

(b) (1.0 pt) Fill in blank (b).

```
print(result)
```

(c) (1.0 pt) Fill in blank (c).

```
if result != "Wait!"
```

(d) (1.0 pt) Fill in blank (d).

```
lambda item_quantity: lambda ordered_at: print(order_meal(item_price*0.50,
item_quantity, ordered_at))
```

(e) (2.0 pt) Rewrite the body of the inner `by` function in a single line of code, such that it behaves the same and passes the same doctests. Note that two calls to `order_meal` with the same parameters will always have the same result.

```
def curry_up_now(item_price):
    def order_quantity(item_quantity):
        def by(ordered_at):
            ----------------------
        return by
    return order_quantity
```

A large text area is provided to give you ample space to write, but your answer must be a valid *single* line of code.

```
print(order_meal(item_price, item_quantity, ordered_at)) or (
(lambda q: lambda h: print(order_meal(item_price*0.50, q, h))) if
order_meal(item_price, item_quantity, ordered_at) != "Wait!" else None)
```

No more questions.